An Image Laboratory

Nielsen, Lars; Elmqvist, Hilding

1983

Document Version:
Publisher's PDF, also known as Version of record

CODEN:LUTFD2/(TFRT-7261)/1-051/(1983)

AN IMAGE LABORATORY

LARS NIELSEN
HILDING ELMQVIST

DEPARTMENT OF AUTOMATIC CONTROL
LUND INSTITUTE OF TECHNOLOGY

SEPTEMBER 1983

LUND INSTITUTE OF TECHNOLOGY
DEPARTMENT OF AUTOMATIC CONTROL
Box 725
S 220 07 Lund 7 Sweden

Document name
Report

Date of issue
September 83

Document number
CODEN:LUTFD2/(TFRT-7261)/1-051/(1983)

Author(s)
Lars Nielsen
Hilding Elmqvist

Supervisor

Sponsoring organization
The Swedish Board of Technical Development

Title and subtitle
An Image Laboratory

Abstract
An Image Laboratory is presented. The purpose is to study how images and image sequences can be used in automatic control applications. The research interest is concentrated on the algorithmic level.

A raster image memory (MATROX) is interfaced to a general computer (VAX-11/780) via a bus interface (UNIBUS-MULTIBUS). Images are grabbed, stored, and presented at a rate of 25 images per second. The image memory is addressable to individual pixels. The support software is such that all user programs are written in PASCAL. Laboratory processes can be controlled by the computer via DA-converters.

It is claimed that this is a simple way to get a comfortable and easy-to-handle system. The speed is limited to the capability of the VAX. However, all solutions of the visual servo problem can in principal be demonstrated.

Key words

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title

ISBN

Language
English

Number of pages
51

Recipient's notes

Security classification

Distribution: The report may be ordered from the Department of Automatic Control or borrowed through the University Library 2, Box 1010, S-221 03 Lund, Sweden, Telex: 33248 lubbis lund.

CONTENTS

1. INTRODUCTION

There has recently been a substantial development in the technology dealing with images and graphics. The main achievements are fast AD-converters and cheaper memory. Consequently hardware for AD-conversion of images, for image storing, and for image presentation, is commercially available. The user does not have to build special hardware for this anymore. Neither is it necessary to do complicated file handling to deal with images.

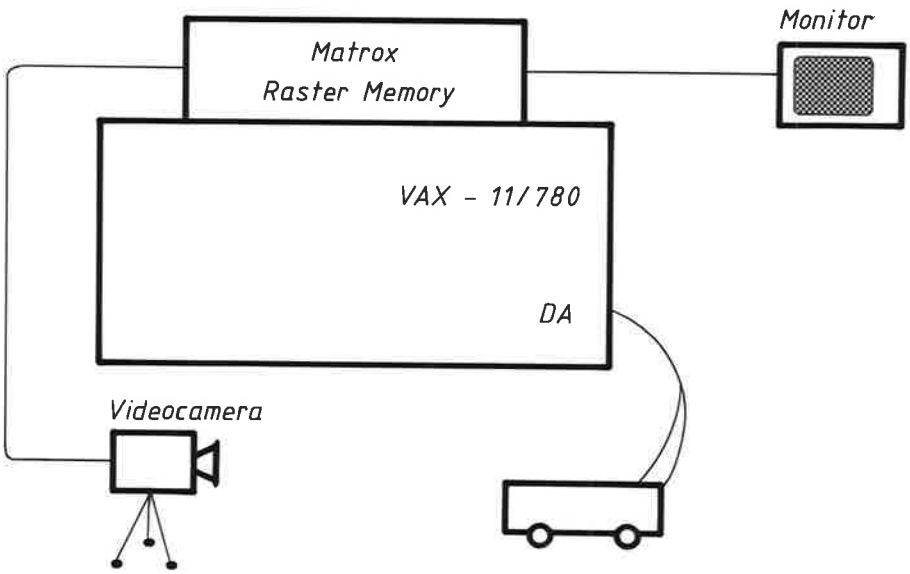
Research on the use of the technology is of significant interest today in the field of automatic control, especially on time sequences of images. With the technology as a background our main interest is thus on the algorithmic level.

A laboratory environment has been created to do experiments. This has meant

- extension of hardware
- software development
- experiments with test and demonstration programs
- definition of laboratory processes for further evaluation of algorithms

The goal has been design and implementation of a convenient and useful system for experiments. To a large extent the purpose is to use standard components to increase the capabilities of the existing VAX based computer system. The level of the basic tools is chosen so that all programming is made in the high level language PASCAL. Moreover there should be good structures for large programs to allow easy accumulation and documentation of the different activities around the system. This report presents the system from the standpoint of a user in image processing.

A schematic overview is presented below.



The main part of the system consists of a Matrox raster image memory interfaced to a VAX-11/780 via a Unibus-Multibus-interface.

The raster memory is 512 x 512 pixels with 8 bits/pixel. For the video output a standard RGB monitor is used. There are four video input channels, where one is used for a video camera. The raster memory has special programmable hardware, which for the input side consists of input channel selection, frame grab commands and setting of gain and offset. This may e. g. be used for hardware thresholding. The memory is addressable to individual pixels. The output hardware consists of video generators (RGB) interpreting the content of the raster memory via a color look up table. This programmable color look up table allows e.g. immediate pseudo coloring and grey scale transformations. Both the input and the output work at the video rate 25 frames per second.

The communication between the raster memory and the VAX is done via the Unibus with a bandwidth of approximately 1.6 MByte/s. The information transferred is both raster register commands and data. Software has been developed to handle this. It consists of procedures for controlling the hardware but also other facilities presented later. Observe that the VAX virtual memory eliminates the need for special code to store matrices of the size 512 x 512. The ADA concept of package is used for the organization of the code. Some laboratory equipments for the experiments are interfaced via DA-converters.

This report presents a simple way to get a comfortable and easy-to-handle system. The speed is limited to the capability of the VAX. However, all solutions of the visual servo problem can in principal be demonstrated. The report is organized as a short presentation in the chapters 2-4. The purpose is to give a feeling for the image laboratory as an environment for control experiments. A lot of essential information for the presumptive user is then placed in the appendices.

2. THE RASTER IMAGE MEMORY

We will concentrate on the raster image memory hardware and its programming. The system consists of different pieces of standard equipment. Building the complete system thus means a couple of interface problems. One problem is that the Matrox hardware is connected to the INTEL Multibus and is not marketed for the VAX Unibus. It is also necessary to map the raster memory registers into the virtual address space of the VAX. This makes it possible to reach the raster memory from a program.

2.1 Hardware

2 Matrox RGB-Graph/64-4

1 Matrox VAF-512

1 Unibus-Multibus-interface

1 Intensa GPC-25 video camera

1 Barco CD 33 HR monitor

The camera is a standard black and white video camera with 1" newvicon. An electronic eye lens with automatic aperture control makes it possible to use the camera in a wide range of light conditions. It is possible to mount other 1" vidicons or other lenses. There is also an external sync possibility.

The Multibus-Unibus-interface was made by Bo Nilsson at the Computer Engineering Department [see Appendix D]. The interface is designed to be transparent. This means the translation from Unibus to Multibus format and vice versa is much faster than the bus bandwidth.

The Matrox system consists of three plug in cards as seen on the photo in figure 2.1.

A block diagram over VAF-512 and RGB-Graph and their connections is seen in figure 2.2.

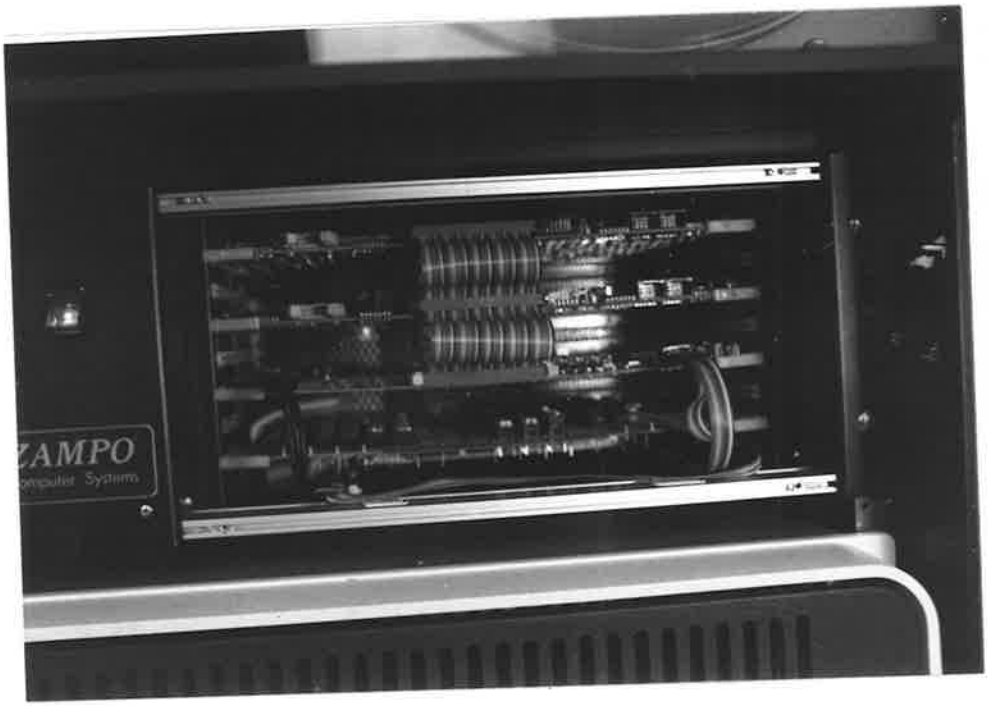


Figure 2.1.

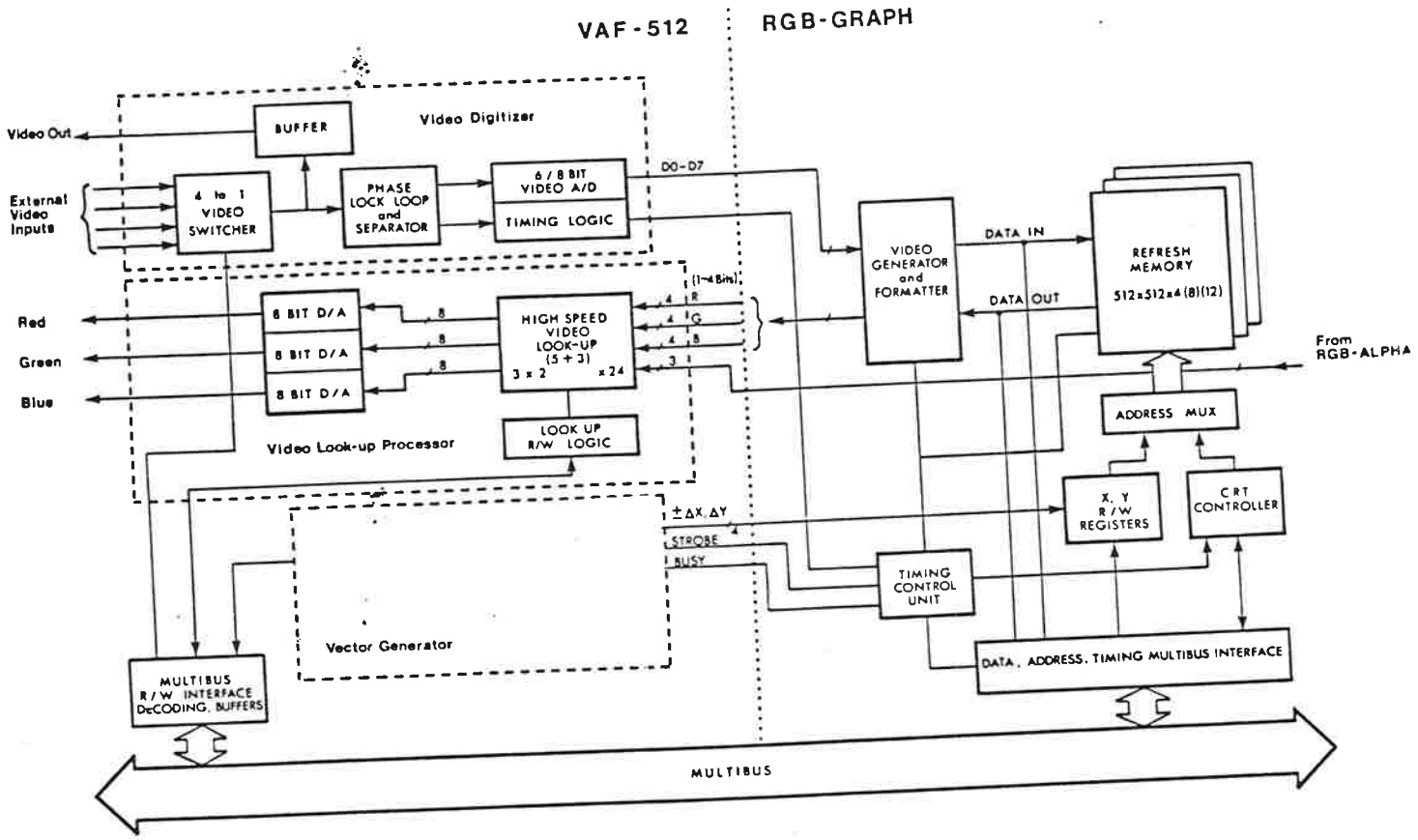


Figure 2.2.

The image system is programmed via commands on bus addresses. The different control registers are presented in Appendix E. This will give a more complete picture of the programming facilities than in the introduction of this report. A full description is found in the manuals. Notice that the 8 bits/pixels are divided into 4 bits on each RGB-Graph card. This means that the system may be viewed as two image planes with 512 x 512 x 4 bits each. Video enable-disable commands may then be used to display one plane while new information is put in the other plane.

2.2 Basic Software - package Raster

The control software between the raster memory and the high level program will be discussed. The code is structured in a package in the way described in chapter 3. The package consists of routines specified in PASCAL and implemented in assembler. The package specification consists of external declarations, and the implementation is put in a library.

There are two main services provided by these routines. One is to map the raster memory registers into the VAX virtual memory. This is done by MapRaster. The other service is to provide communication procedures. Among these there are both general procedures and more special e.g. those handling the raster memory as two image planes.

The specification of the package Raster in Appendix B gives a detailed presentation of the low level software.

3. SOFTWARE DEVELOPMENT ENVIRONMENT

The focus will now be on the user situation. The software system is supposed to be easy to use. To achieve this, PASCAL procedures and data are grouped together using the concept of package from ADA.

The user is supposed to do two things. First write a PASCAL program as described in the following. Here may any procedure from any package be used. Second write a compile and link command procedure. This command procedure shall define the used packages to the preprocessor. Furthermore the link command shall invoke the used libraries. The example in Appendix A illustrates these subjects.

The packages

Our packages are text files. A package is divided into sections by preprocessor commands. The section keywords are

```
.PROGRAM        .FORWARD
.LABEL          .PROCEDURE
.CONST          .INIT
.TYPE           .MAIN
.VAR            .END
.EXTERN
```

The commands defines sections of a Pascal program. The .INIT command defines a section that will be placed first in the main body. The .END command results in that the preprocessor neglects the text until next command. It is possible to specify the sections in any order i.e. to mix .TYPE, .VAR, .PROCEDURE sections.

This is the heart of the software development. The users write their own packages where one is the main program, and may use any of the other packages. This modularization makes it easier for several implementors to work together in the same project. It is also easy to accumulate finished parts of a project in a package.

We have structured our packages in the spirit of ADA with a specification part and a body part. The specification part contains what is usable outside the package, and the body contains the implementation. To do this in PASCAL the procedure specification is separated from its code using forward declarations in a starting .FORWARD section.

Support packages

There are a number of support packages implemented in PASCAL. The specification part of some of these are presented in appendix B.

The preprocessor ComPack

The preprocessor ComPack is a file handling program used to combine packages. The input to the program is a list of files of package type. The sections of each package file are appended to files corresponding to the section types. A sceleton program compackprog includes the section files. A number of garbage files with extension .sec are generated and should be deleted.

The command procedure

We will concentrate on the example demo.com in appendix A. The procedure is written in the VAX control language DCL. A call is made as

```
$ @demo
```

This will result in an executable program demo.exe in the following way.

Line 8 DCL symbol starting compack.

Line 9-14 List of packages used.

Line 17 Compile the Pascal program with the name compackprog.

Line 20 Link the program. Invoke the library Raster.

Line 22-23 Clean up.

4. EXPERIMENTAL SETUP

We have some experimental setups at the department. They are used for control experiments in the lab. Automatic control of dynamical systems uses feedback from the process variables. If a camera is used the control is based on a pattern in an image, a position of an object, recognizing a shape etc.

To study such control we have interfaced the lab processes to the VAX. Standard DA-converters are used. The software needed to include this in the software system is placed in the support package ADconv.pak. Primitives for handling real time programming in PASCAL are developed. To handle the communication between concurrent processes the rendezvous concept from ADA is used (see appendix C). The photo below illustrates one setup. The car is controlled using only image information.

Another example is the Turtle seen below. The main difference is that the Turtle has touch-sensors, indicating if it hits an object. The control is based on a combination of image and touch information.

foto
orginalet finns i
TFRT - 5271

The camera may also easily be used to supervise other processes. Examples are a tank process, a ball and beam process or a minimover MICROBOT.

5. ACKNOWLEDGEMENTS

Lars Nielsen, Hilding Elmqvist, and Tommy Essebo have developed the system in collaboration. All three were supported by the Swedish Board of Technical Development. Hilding Elmquist and Tommy Essebo were in the project "Development of User Oriented Languages and Software Tools for Implementation of Control Systems" (STU-80-3962d). Lars Nielsen was in the project "Control Based on Image Information" (STU-82-3429). The Department of Automatic Control in Lund is also acknowledged for its supporting skills and knowledge.

6. REFERENCES

RGB-Graph Color Graphics Controller, Manual no. 167MO-04-0, Matrox Electronic Systems

VAF-512 Graphics Support Board, Manual no. 176MO-01-0, Matrox Electronic Systems

Tengvall Freddy, Rendezvous Primitives for Intertask Communication on VAX/VMS. CODEN:LUTFD2/(TFRT-7234)/1-32/(1982)

APPENDIX A

User Example

A complete user program is presented. Both the program and the command procedure are listed.

First a command procedure called demo.com is presented. The packages mentioned below and demo.pak are listed in demo.com as input to compack. The assembler routine library is invoked in the link command.

Second a Pascal program called demo.pak is presented. It is structured as a package using the section keywords. The program uses type declarations, external declared procedures and Pascal procedures from the packages Raster.pak, RasterReg.pak, LookUp.pak, ImageDef.pak, and SimplFil.pak (compare Appendix B).

Remark 1 More information is found in chapter 3.

Remark 2 The program modification cycle is
1. Edit demo.pak
2. @demo

```
! Command procedure to generate the DEMO program

$ write sys$output " "
$ write sys$output "Generating DEMO program."
$ write sys$output " "
$ write sys$output "Handling packages."
$ show time
$ compack
Raster.pak
RasterReg.pak
LookUp.pak
ImageDef.pak
SimplFil.pak
Demo.pak
$ write sys$output "Compiling."
$ show time
$ pascal/object=demo compackprog
$ write sys$output "Linking."
$ show time
$ link/nomap demo,rasteropt/opt
$ show time
$ delete demo.obj;*
$ delete *.sec;*
```

```
package DEMO is

-- A demonstration program

-- Author: Lars Nielsen
-- Date: 1983-09-07

------------------------------------------------------------

package DEMO body is

.PROGRAM

program demo(input,output,imagefile);

.TYPE

optype          = array[-1..1,-1..1] of real;

operatortype    = record
  op            : optype;
  opsum         : real;
end;

.VAR

exit                        : boolean;
comchar                     : char;
operator                    : operatortype;
inimage, outimage           : imagetype;    { from ImageDef.pak }

.PROCEDURE

{-------------------------------------------------------------}

function limit(val: integer): integer;
  const minI = 0; maxI = 255;
  begin
  if val > maxI then limit:=maxI
  else if val < minI then limit:=minI
  else limit:=val;
  end;

{-------------------------------------------------------------}

procedure Threshold(level: integer;
                    var Inimage, OutImage: imagetype);
  { threshold InImage, put result in OutImage }
  const minx = 1; maxx = 512; miny = 1; maxy = 512;
        minI = 0; maxI = 255;
  var x,y: integer;
  begin
  for y := miny to maxy do for x:= minx to maxx do
  if InImage[y,x] > level then OutImage[y,x] := maxI
  else OutImage[y,x] := minI;
```

```
  end;

{-------------------------------------------------------------}

procedure Invert(var Inimage, OutImage: imagetype);
  { Invert InImage, put result in OutImage }
  const minx = 1; maxx = 512; miny = 1; maxy = 512;
        minI = 0; maxI = 255;
  var x,y: integer;
  begin
  for y := miny to maxy do for x:= minx to maxx do
  OutImage[y,x] := maxI - InImage[y,x];
  end;

{-------------------------------------------------------------}

procedure NewOperator;
  { define new 3 x 3 neighbourhood operator }
  var i,j : integer;
  begin
  with operator do
    begin
    for j:=-1 to 1 do for i:= -1 to 1 do
      begin
      write('operator[ ',j:2,', ',i:2,'] := ');
      readln(op[ j, i]);
      end;
    opsum:=0;
    for j:=-1 to 1 do for i:= -1 to 1 do
      opsum:=opsum+op[j,i];
    end;
  end;

{-------------------------------------------------------------}

procedure LowPassOperator;
  begin
  with operator do
    begin
    op[-1,-1] := 1;  op[-1, 0] := 1;  op[-1, 1] := 1;
    op[ 0,-1] := 1;  op[ 0, 0] := 2;  op[ 0, 1] := 1;
    op[ 1,-1] := 1;  op[ 1, 0] := 1;  op[ 1, 1] := 1;
    opsum := 10;
    end;
  end;

{-------------------------------------------------------------}

procedure HighPassOperator;
  begin
  with operator do
    begin
    op[-1,-1] := -1;  op[-1, 0] := -1;  op[-1, 1] := -1;
    op[ 0,-1] := -1;  op[ 0, 0] :=  9;  op[ 0, 1] := -1;
    op[ 1,-1] := -1;  op[ 1, 0] := -1;  op[ 1, 1] := -1;
```

```
    opsum := 1;
    end;
  end;

{-------------------------------------------------------------}

procedure Operate(var InImage,OutImage: imagetype);
  { apply operator on InImage, put result in OutImage }
  const minx = 2; maxx = 511; miny = 2; maxy = 511;
  var x,y,i,j,help: integer;
      temp        : real;
      localop     : optype;
      localopsum  : real;

  begin
  localop:=operator.op;
  localopsum:=operator.opsum;
  for y := miny to maxy do for x:= minx to maxx do
    begin
    temp:=0;
    for j:=-1 to 1 do for i:=-1 to 1 do
      begin
      help:=inimage[y+j,x+i];
      temp:=help*localop[j,i]+temp;
      end;
    outimage[y,x]:=limit(round(temp/localopsum));
    end;
  end;

{-------------------------------------------------------------}

procedure Init;
  var x,y: integer;
  begin
  LowPassOperator;
  for x:= 1 to 512 do for y:= 1 to 512 do
    begin
    inimage[ x, y] := 0;
    outimage[ x, y] := 0;
    end;
  end;

{-------------------------------------------------------------}

.MAIN

Init;
exit := false;
repeat
  writeln;
  writeln('------- Menu -------------------------------');
  writeln;
  writeln('E[xit],F[reeze],G[rabCont],S[ave],R[estore]');
  writeln;
  writeln('B[lackWhite],P[seudoColor]');
```

```
    writeln;
    writeln('T[hreshold], I[nvert]');
    writeln;
    writeln('N[ewoperator], O[perate], L[owpass], H[ighpass]');
    writeln;
    write('Command > ');

    readln(comchar);
    if comchar in ['e', 'E', 'f', 'F', 'g', 'G',
                   's', 'S', 'r', 'R',
                   'b', 'B', 'p', 'P',
                   't', 'T', 'i', 'I',
                   'n', 'N', 'o', 'O', 'l', 'L', 'h', 'H'] then
      case comchar of
        'e','E' :   exit := true;
        'f','F' :   begin
                    GrabOne;                { from RasterReg.pak }
                    ReadImage(InImage);     { from ImageDef.pak }
                    end;
        'g','G' :   GrabCont;               { from RasterReg.pak }
        's','S' :   SaveFile(InImage);      { from SimplFil.pak }
        'r','R' :   begin
                    RestoreFile(InImage); { from SimplFil.pak }
                    WriteImage(InImage);  { from ImageDef.pak }
                    end;
        'b','B' :   BlackWhite256;            { from LookUp.pak }
        'p','P' :   Pseudo;                   { from LookUp.pak }
        't','T' :   begin
                    Threshold(128, InImage, InImage);
                    WriteImage(InImage);
                    end;
        'i','I' :   begin
                    Invert(InImage, InImage);
                    WriteImage(InImage);
                    end;
        'n','N' :   NewOperator;
        'o','O' :   begin
                    Operate(InImage, OutImage);
                    WriteImage(OutImage);
                    InImage := OutImage;
                    end;
        'l','L' :   LowPassOperator;
        'h','H' :   HighPassOperator;
      end
    else writeln('Error in command');
  until exit;

.END
```

APPENDIX B

Packages

The specification part of some support packages are listed. They are

Raster.pak

RasterReg.pak

LookUp.pak

ImageDef.pak

ADConv.pak

DigConv.pak

SimplFil.pak

The package Raster is implemented in assembler. All other packages are implemented in PASCAL.

```
------------------------------------------------------------

package RASTER is

-- Procedures for Raster Operations

-- Authors: Tommy Essebo, Lars Nielsen
-- Date: 1983-09-06

-- If the logical name NORAST is defined when MapRaster is
-- called the actual connection to the raster hardware is
-- inhibited.

-- If the logical name RASTCHECK is defined when MapRaster is
-- called a parameter check is made in various procedure
-- calls and a diagnostic printout occurs if an error is
-- found.

------------------------------------------------------------

.TYPE

bytes = 0..255;

.EXTERN

function MapRaster: integer; extern;
  { Initializes procedures. This procedure handles the
    mapping of registers into the virtual memory. A call to
    this procedure should be done as the first thing in a
    user program.  This results in a call to MapIOPage which
    in turn will call a System Services routine. This will
    map the virtual addresses of the variables in the library
    to a physical VAX iopage address. Notice that the user
    needs not to be priviliged to use it, since MapIOPage is
    installed on the VAX system with priviliges. }

procedure MultiSet(nr: integer; word: integer); extern;
  { Write on Multibus address nr.  MultiSet and MultiGet are
    the fundamental operations to be able to read and write
    on any Multibus address. This means that all other
    operations may be expressed in these, but are though
    useful either of convenience or speed considerations. }

procedure MultiGet(nr: integer; var word: integer); extern;
  { Read on Multibus address nr. }

procedure SetRasterReg(nr: integer; word: integer); extern;
  { Write on address nr in active plane. }

procedure GetRasterReg(nr: integer;var word: integer); extern;
  { Read on address nr in active plane. }

{------------------------------------------------------------}
```

```
procedure RasterRead(var image: array [n1..n2: integer] of
                packed array[m1..m2: integer] of bytes;
                xc, yc, nx, ny, increment: integer); extern;
  { Reads area (xc, yc) , (xc+nx, yc+ny) from Raster memory
    to the variable image.
    Note that (xc,yc) is upper left corner of the area and
    that (xc+nx,yc+ny) is lower right corner.
    (xc,yc) will be stored in image[ 1, 1].}

procedure RasterWrite(var image: array [n1..n2: integer] of
                packed array[m1..m2: integer] of bytes;
                xc, yc, nx, ny, increment: integer); extern;
  { Writes area (xc, yc) , (xc+nx, yc+ny) to Raster memory
    from the variable image.
    Note that (xc,yc) is upper left corner of the area and
    that (xc+nx,yc+ny) is lower right corner.
    (xc,yc) will be fetched from image[ 1, 1].}

procedure RasterMS4Read(var image: array [n1..n2: integer] of
                packed array[m1..m2: integer] of bytes;
                xc, yc, nx, ny, increment: integer); extern;
  { Same as RasterRead but only Most Significant 4 bits. }

procedure RasterMS4Write(var image: array [n1..n2: integer] of
                packed array[m1..m2: integer] of bytes;
                xc, yc, nx, ny, increment: integer); extern;
  { Same as RasterWrite but only Most Significant 4 bits. }

{-------------------------------------------------------------}

procedure VisiblePlane(nr: integer; visible: boolean); extern;
  { Set plane visibility status. }

procedure DrawPlane(nr: integer); extern;
  { Enables drawing in plane nr. }

procedure RasterErase(color:integer); extern;
  { Clears active plane to value color. }

procedure RasterCol(color:integer); extern;
  { Specify a new color. }

function ReadPixel(x, y: integer): integer; extern;
  { Get value at (x,y). }

procedure WritePixel(x, y: integer); extern;
  { Write a dot of current color in (x,y). }

{-------------------------------------------------------------}

procedure RasterClip(x1,y1,x2,y2: integer); extern;
  { Set limits for VerLine and HorLine. }

procedure RasterVerLine(x1,y1,y2:integer); extern;
  { Draw vertical line. }
```

```
procedure RasterHorLine(y1,x1,x2:integer); extern;
  { Draw horizontal line. }

procedure RasterPaint(xlow,ylow,xhigh,yhigh:integer); extern;
  { Paint a window on screen in current color. }

.END
```

```
-------------------------------------------------------------

package RasterReg is

-- Procedures to initialize and handle the Matrox registers
-- Reference: Matrox VAF-512 manual page 24

-- Author: Lars Nielsen
-- Date: 1983-09-06

-------------------------------------------------------------

.FORWARD

procedure GrabOne; forward;
  { Grab one frame and freeze it. }

procedure GrabCont; forward;
  { Do continuous frame grabbing to Raster memory. }

.END

-------------------------------------------------------------


-------------------------------------------------------------

package LookUp is

-- procedures to program the look up table

-- Authors: Hilding Elmqvist, Lars Nielsen
-- Date: 1983-09-06

The look up table has 256 entries numbered 0 .. 255.
Each entry has an eight bit register for each of the
colors r, g and b.

-------------------------------------------------------------

.FORWARD

procedure SetColorMap(color, r, g, b: integer); forward;
  { Set r g b values in entry color }

procedure GetColorMap(color:integer;var r,g,b:integer); forward;
  { Get r g b values from entry color }

procedure BlackWhite16; forward;
  { Initializes color look up table for 16 uniformly
    distributed intensities of black and white. }

procedure BlackWhite256; forward;
  { Initializes color look up table for 256 uniformly
    distributed intensities of black and white. }
```

```
procedure TwoPlanes; forward;
  { Initializes color look up table for
    two planes of 16 colors. }

procedure Pseudo; forward;
  { Initializes color look up table for 16 pseudo colors. }

procedure ColorPlane; forward;
  { Initializes color look up table for
    one plane of 256 colors. }

.END
```

```
package IMAGEDEF is

-- Definitions for image handling using RASTER routines

-- Author: Tommy Essebo
-- Date: 1982-11-02


The raster memory is represented by a matrix of bytes:
image[row, column] , where 1 <= row, coloumn <= 512
The top left pixel is image[1,1] and lower left corner
is in image[512, 1]
```

```
.CONST

imagesize = 512;

.TYPE

imageline = packed array [1..imagesize] of bytes;

imagetype = array [1..imagesize] of imageline;

.FORWARD

procedure ReadImage(var image: imagetype); forward;
  { Reads one image from Raster memory }

procedure WriteImage(var image: imagetype); forward;
  { Writes one image to Raster memory }

.END
```

```
------------------------------------------------------------

package ADCONV is

-- procedures for A/D and D/A conversion

-- Author: Tommy Essebo
-- Date: 1983-01-27


The analog values are real numbers in the range -1 to 1
A/D input  channels: 0 - 15
D/A output channels: 0 - 5
AnalogOut will limit the values if needed

------------------------------------------------------------

.FORWARD

function AnalogIn(chan: integer): real; forward;

procedure AnalogOut(chan: integer; val: real); forward;

.END

------------------------------------------------------------


------------------------------------------------------------

package DIGCONV is

-- procedures for digital input/output

-- Author: Tommy Essebo
-- Date: 1983-02-09

The digital values are boolean values
Input  channels: 0 - 15
Output channels: 0 - 15

------------------------------------------------------------

.FORWARD


function DigitalIn(chan: integer): boolean; forward;

procedure DigitalOut(chan: integer; val: boolean); forward;

.END

------------------------------------------------------------
```

```
------------------------------------------------------------

package SIMPLEFILER is

-- Procedures to save and restore imagefiles.
-- The  image is 512x512 bytes.

-- Author: Lars Nielsen
-- Date: 1983-09-06

------------------------------------------------------------

.TYPE

filenametype       = packed array [1..60] of char;

.VAR

imagefile                  : file of imageline;

.FORWARD

procedure Restorefile(var image: imagetype); forward;
  { Asks for a filename and reads it to
    the Pascal matrix image. }

procedure Savefile(var image: imagetype); forward;
  { Asks for a filename. Writes the Pascalmatrix image
    to this file. }

.END

------------------------------------------------------------
```

APPENDIX C

Rendezvous in Pascal

Abstract

The ADA rendezvous mechanism for inter-process communication and synchronization is implemented for Pascal programs using procedure calls and records with variants.

Rendez-vous concepts

The following ADA concepts for communication and synchronization between tasks are implemented:

Entry call	Accept call # 1	Accept call # 2

<entry call>	select
	  <accept1>
	  .
	or
	  <accept2>
	  .
	or
	  <acceptn>
	  .
	or
	  delay <sec>
	  .
	end select
		select
		  <accept1>
		  .
		or
		  <accept2>
		  .
		or
		  <acceptn>
		  .
		else
		  .
		  .
		end select

Entries and processes.

All entries declared in one task forms a group qualified by the process. The intertask message format is related only to the entry since this is used as tag-field for the variant part of the message record.

Task declaration and activation.

In Ada a task is activated when the declaration of it is executed. In this implementation a task must be explicitly activated by a DCL command (usually SPAWN). A task is an executable image in the VMS environment and it must be created by the usual compile and link sequence before it is activated. A task is terminated when all it's statements are executed (or when an exception occurs).

Description of individual procedures/primitives.

InitRV(sectname, filename: packed array [integer] of char);

Initializes the rendezvous package.
sectname - name of the global section to be used for
communication between the tasks.
filename - name of the section file (usually a logical name).

procedure DeclareEntries(process: processtype; validentries: entryset);

Declares all entries in one task.
process - process to be declared
validentries - set of entries in this process

procedure ConnectEntries(process: processtype);

Declares a process that can subsequently be called from this task. If the process is not yet declared by another task, the calling task will wait at this point until this is done before continuing.

function EntryPar: pmessagetype;

Returns pointer to a global message record to be used in all entry calls in one process.

procedure CallEntry(process: processtype; par: pmessagetype);

Makes an entry call.
The specified entry in the process must be set in the message. par must point to the global message record retrieved from EntryPar. The answer (if any) from the rendezvous is returned in the same record as the call.

procedure SelectAccept(acceptset: entryset; var par: pmessagetype; delay: real);

Makes an accept call.
At the call par should point to nil since a new value will be returned by SelectAccept.
delay - timeout value in seconds. If delay = 0 there is no timeout (i. e. delay is infinite) and if delay < 0 the else construct of the select statement will be used. In this case the SelectAccept call will return immediately if there is no entry call waiting. Par will be set to nil if no rendezvous was possible because of else or timeout.

procedure EndAccept(var par: pmessagetype);

Ends the accept statement.
Par will be set to nil by this call.

Message format

All messages consist of a fixed part and a variable user-defined part implemented as a record with variants. The fixed part of the messages consists of the following fields:
fp, bp: pointer;
processpid: integer;
returnaddress: pointer;
entry: entries;
Fp, bp, processpid and returnaddress are set by the rendezvous procedures and should not be changed by the user. The last field, entry is used as case tag-field for the variant part. It must be set by the user when making an entry call to indicate which entry in the process to be called.

Example:

```
{ processA }
 .
var   p: pmessagetype;
 .
 .
InitRv('SECT', 'SECTFILE');
 .
 .
ConnectEntries(processB);
 .
p := EntryPar;
while true do
  begin
  { Define message }
  p^.entry := entry1;
  p^.data := ....
   .
  CallEntry(processB, p);
   .
   .
   .
   .
   .
   .
   .
   .
  { check answer }
  with p^. do ....
  end { while true };
```

```
{ processB }

var   p: pmessagetype;
      acceptset: entryset;
   .
InitRv('SECT', 'SECTFILE');
   .
acceptset := [entry1, entry2];
DeclareEntries(processB, acceptset);
   .
   .
while true do
  begin
    .
    .
    .
    .
  SelectAccept(acceptset, p, 5.0);
  if p = nil then { timeout case }
  else { perform accept code }
  case p^.entry of
  entry1: ....
  entry2: ...
  end { case };
    .
  EndAccept(p);
     .
     .
  end { while true };
```

APPENDIX D

Unibus to Multibus converter

A short excerpt from the report of Bo Nilsson. The block diagram gives the idea. The lay out on the two interface cards are presented. One card is installed on the Unibus and the other card on the Multibus.

UNIBUS TO MULTIBUS CONVERTER

BO NILSSON

EDT, LTH

LUND 820503

Allmänt.

Read modify write, DMA och avbrott är ej implementerat. Vad gäller RMW torde detta ej innebära någon inskränkning. Detta med tanke på att kort som ansluts till Multibus måste vara slavkort. En möjlighet att koppla ur interfacekortet till Multibus finns emellertid tillgänglig. En yttre signal $\overline{\text{BREQ}}$ aktiveras, vilket resulterar i att signalen $\overline{\text{BACK}}$ (synkroniserad med $\overline{\text{MSYN}}$) talar om att Multibus är ledig. En mer välordnad synkronisering är tänkt att skötas via en seriekanal.

Vid urkoppling av spänning till Multibus-racket är det lämpligt att ställa omkopplaren på interfacekortet till Unibus i läge off.

Vid eventuell omkonstruktion kan receiverkretsarna (8640) och driverkretsarna (8881) bytas ut mot tranceiverkretsar (8641), vilket ger dubbelriktad anslutning till Unibus. Med ytterligare logik kan sen DMA och avbrott troligen implementeras relativt enkelt.

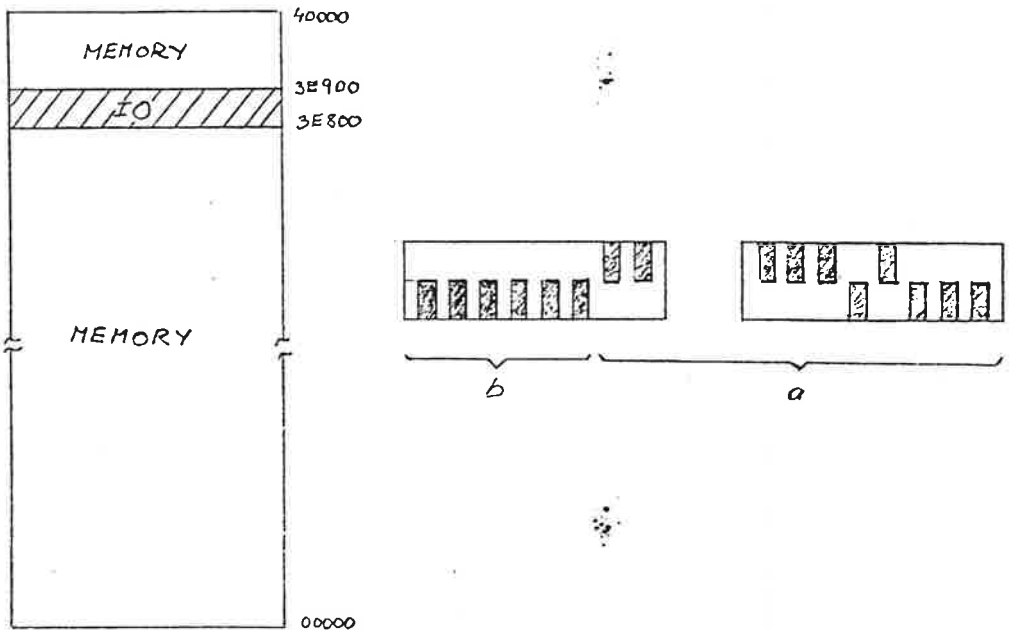
Inställning av dip-switchar.

På Multibus är minne och IO separerade. Eftersom Unibus endast har minnes-adressering är det nödvändigt att avsätta en del av Unibus adressrymd till IO. Med de 10 lsb på dipswitcharna avkodas 256 byte till IO. Adresser utanför dessa 256 byte blir minne. På de kort som placeras i Multibus-racket måste därför minne och IO ligga i olika adressområden. Observera att om Multibus-kort med 16 bitars IO-adress används, måste avkodningen för de 8 msb på IO-kortet stämma överens med inställningen på dipswitcharna.

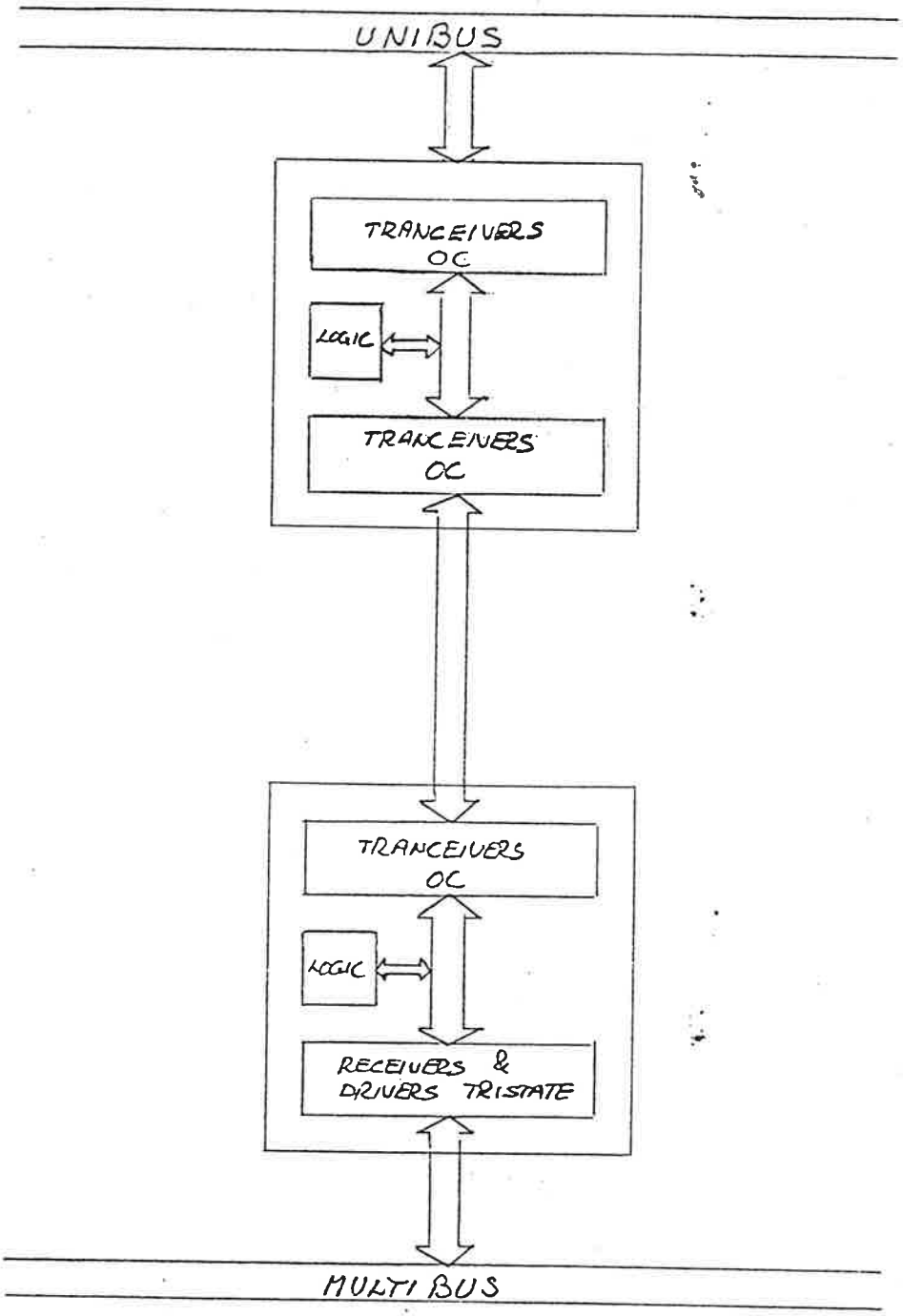
Eftersom antalet adressledningar på Multibus är 24 och på Unibus endast 18 måste de 6 msb tvångssättas. Detta görs med de 6 msb på den vänstra dipswitchen.

Ex: a) adress 3E800 - 3E8FF avsätts till IO

b) alla kort i Multibus-racket har minnesadresser lägre än 03FFFF



BLOCK-SCHEMA



UPPER CONNECTOR

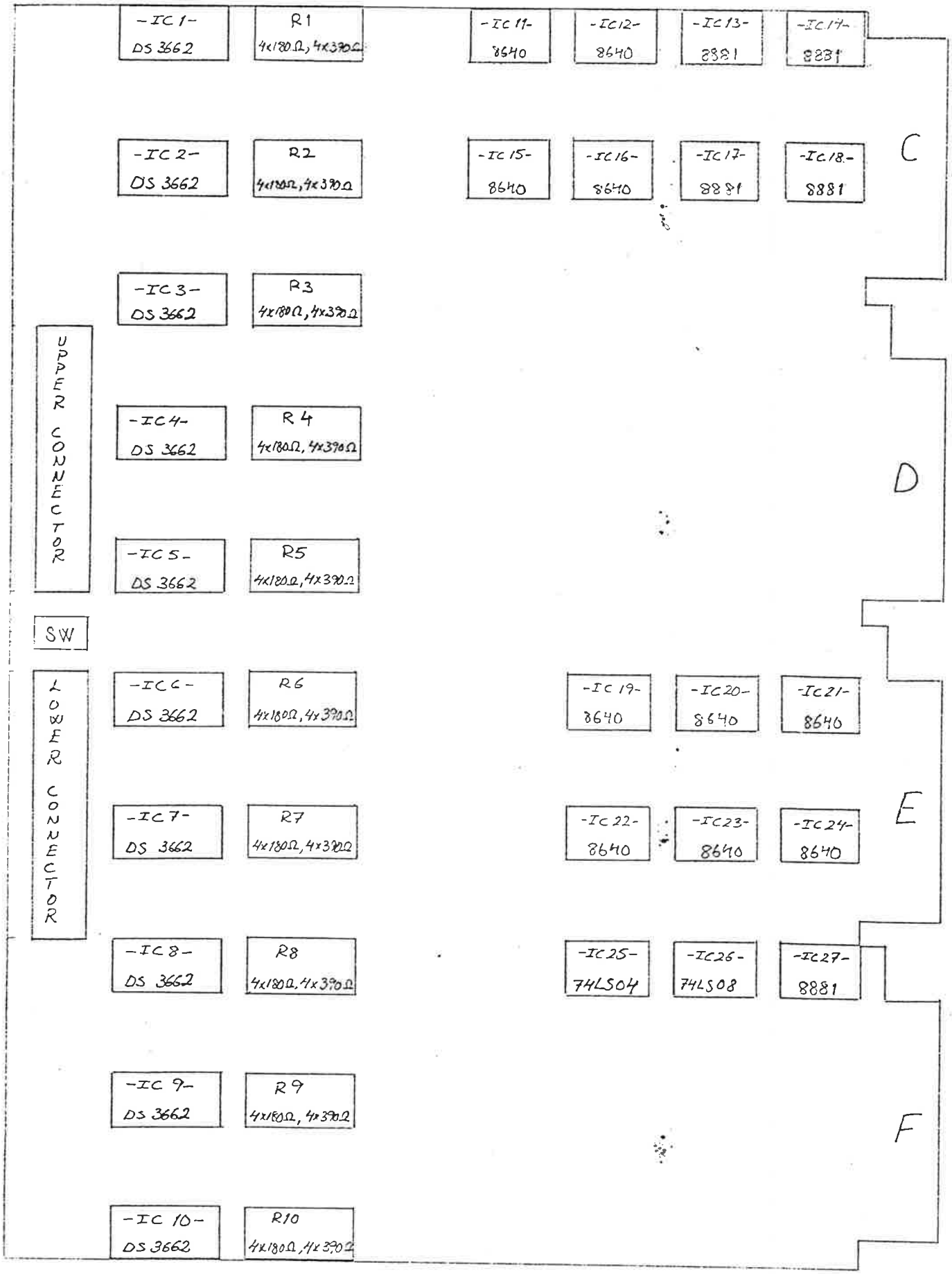
LOWER CONNECTOR

P2

-IC 1- DS3662	-IC 2- DS3662	-IC 3- DS3662	-IC 4- DS3662	-IC 5- DS3662
-IC 6- DS3662	-IC 7- DS3662	-IC 8- DS3662	-IC 9- DS3662	-IC 10- DS3662
R1 8×390Ω	R2 8×390Ω	R3 8×390Ω	R4 8×390Ω	R5 8×390Ω
R6 8×180Ω	R7 8×180Ω	R8 8×180Ω	R9 8×180Ω	R10 8×180Ω

P1

R11 15×1KΩ	-IC 11- 74LS136		-IC 12- 74LS240	-IC 13- 74LS240
R12 15×1KΩ	-IC 14- 74LS136	-IC 15- 74LS136	-IC 16- 74LS240	-IC 17- 74LS240
DIPSW 1	DIPSW 2		-IC 18- 74LS240	-IC 19- 74LS240
				-IC 20- 74LS240
	-IC 21- 74LS04	-IC 22- 74LS04	-IC 23- 74LS00	-IC 24- 74LS240
	-IC 25- 74LS03	-IC 26- 74LS08	-IC 27- 74LS32	-IC 28- 74LS240
		-IC 29- 74LS74	-IC 30- 74LS138	-IC 31- 74LS244


-IC 1-
DS 3662
R1
4x180Ω, 4x390Ω
-IC 11-
8640
-IC12-
8640
-IC13-
8881
-IC14-
8881
C
-IC 2-
DS 3662
R2
4x180Ω, 4x390Ω
-IC 15-
8640
-IC 16-
8640
-IC17-
8881
-IC18-
8881
-IC 3-
DS 3662
R3
4x180Ω, 4x390Ω
UPPER CONNECTOR
-IC4-
DS 3662
R4
4x180Ω, 4x390Ω
D
-IC 5-
DS 3662
R5
4x180Ω, 4x390Ω
SW
LOWER CONNECTOR
-IC 6-
DS 3662
R6
4x180Ω, 4x390Ω
-IC 19-
8640
-IC20-
8640
-IC21-
8640
-IC 7-
DS 3662
R7
4x180Ω, 4x390Ω
-IC 22-
8640
-IC23-
8640
-IC 24-
8640
E
-IC 8-
DS 3662
R8
4x180Ω, 4x390Ω
-IC 25-
74LS04
-IC26-
74LS08
-IC27-
8881
-IC 9-
DS 3662
R9
4x180Ω, 4x390Ω
F
-IC 10-
DS 3662
R10
4x180Ω, 4x390Ω

APPENDIX E

Raster Registers

The set of registers in the Matrox system is presented. This gives a complete picture of the programming possibilities. The first 7 pages are from the Matrox VAF-512 manual. The next 8 pages are from the Matrox RGB-Graph/64-4 manual.

Technical note

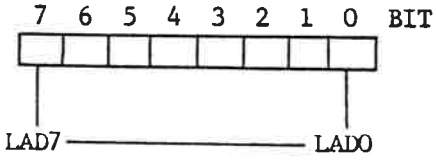
The two RGB-Graphs are placed on different Multibus addresses. Then on the slave card the strap 72-71 shall be out and strap 73-72 be in.

4.0 REGISTERS:

The VAF-512 is programmed via a series of registers and ports that are described in subsections 4.1 through 4.12. These locations are all accessed by programmed I/O and can be strapped on any 16 address boundary within system I/O space.

4.1 ADDRESS REGISTER LOW:

WRITE ONLY



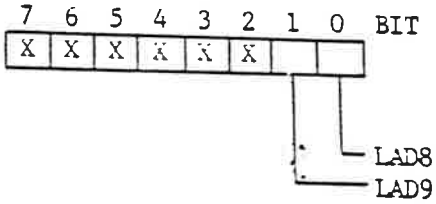
BASE ADDRESS + 00H = LOC

AS-SHIPPED ADDRESS = 60H

LAD0 through LAD7 are the lower 8 bits of look-up table address. The address minus one of the look-up table location to be accessed is first loaded into Address Register Low and Address Register High, then the location is accessed through one of the three data ports. The address registers are automatically incremented before each data port access, so the Address Register need only be accessed once when a look-up table is being filled.

4.2 ADDRESS REGISTER HIGH:

WRITE ONLY



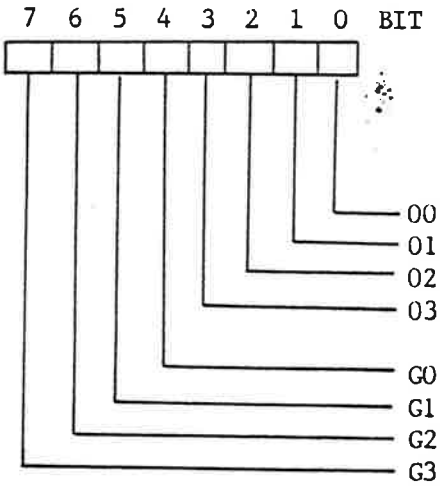
BASE ADDRESS + 01H = LOC

AS-SHIPPED ADDRESS = 61H

LAD8 and LAD9 are the two most significant bits of the look-up table address.

4.3 GAIN/OFFSET REGISTER:

WRITE ONLY



LOC

BASE ADDRESS + 02H = ☐

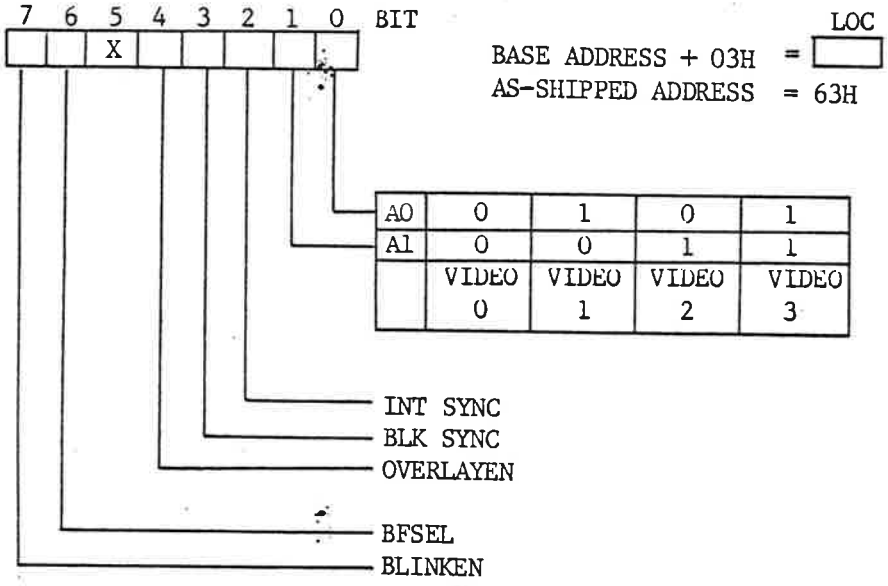
AS-SHIPPED ADDRESS = 62H

BITS 0-3: 00-03. These four bits provide digital control of the offset of the analogue video signal with respect to the ADC. 0H written to these bits produces zero offset. That is to say. the black level of the analogue signal will generate 00H at the output of the ADC. FH written to these bits will offset the analogue signal half of full scale. That is to say, a level halfway between black and white will cause the ADC to generate 00H. Sixteen offset levels are possible. This feature is used in conjunction with the gain control to expand sections of the analogue video signal.

BITS 4-7: G0-G3. These four bits provide digital control of the ADC gain. FH written to these bits produces normal gain: if there is no offset, the whole analogue signal will be digitized with the black level producing 00H and the white level producing FFH or 0FH depending on which ADC is used. If 0H is written to these bits, the gain is doubled and only half of the analogue signal will be digitized. In effect, this allows half of the signal to be expanded to full scale. The offset bits can be used to position the expanded section within the full scale. Sixteen levels of gain can be set.

4.4 MODE REGISTER:

WRITE ONLY



BITS 0 AND 1: A0 AND A1. The user uses these two bits to enable one of the four video inputs to the Frame Grabber.

BIT 2: INT SYNC. When a one is written to this bit, the PLL will use the on-board sync source. When a zero is written to this bit, the PLL will lock onto the sync in the video signal that the user supplies at the selected video input. When using Frame Grab, external sync must be used. When changing from Internal Sync to External Sync or vice versa, leave enough time (abt .1 sec) for internal timing to stabilize.

BIT 3: BLK SYNC. When a one is written to this bit, the Frame Grabber's PLL is programmed to lock onto block sync provided at the selected video input. When a zero is written to this bit, the PLL is programmed to lock onto serrated sync (CCIR/EIA). This bit should be zero if Internal Sync is used.

BIT 4: OVERLAYEN: When a one is written to this bit, an RGB-Alpha can insert text on the display. When this bit is zero the RGB-Alpha cannot insert text on the display. This bit must be zero if no RGB-Alpha is used.

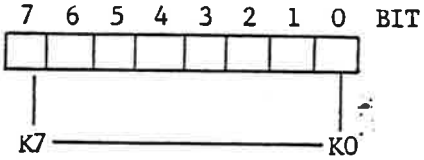
4.4 MODE REGISTER (cont'd):

BIT 6: BFSEL: When a one is written to this bit, the blink frequency is programmed to be 3.75 Hz. When a zero is written to this bit, the blink frequency is programmed to be 1.8Hz.

BIT 7: BLINKEN: When a 1 is written to this bit, Blink is enabled. When this bit is zero, Blink is disabled.

4.5 VECTOR SLOPE REGISTER:

WRITE ONLY



LOC

BASE ADDRESS + 04H =

AS-SHIPPED ADDRESS = 64H

K0 through K7 are the lower 8 bits of the vector slope (K), which is a 10 bit binary value determined as follows:

$$K = \frac{|\Delta Y|}{|\Delta X|} \text{ when } |\Delta X| \geq |\Delta Y|$$

OR

$$K = \frac{|\Delta X|}{|\Delta Y|} \text{ when } |\Delta X| < |\Delta Y|$$

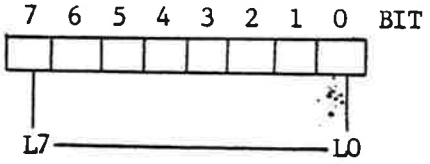
The value of K is always equal to or less than 1 and is loaded into K0-K9 as follows:

2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}	2^{-7}	2^{-8}	2^{-9}.
K9	K8	K7	K7	K5	K4	K3	K2	K1	K0

The two most significant bits of the slope (K8, K9) are placed in the Start Register.

4.6 X-Y LENGTH REGISTER:

WRITE ONLY



BASE ADDRESS + 05H = ☐
AS-SHIPPED ADDRESS = 65H

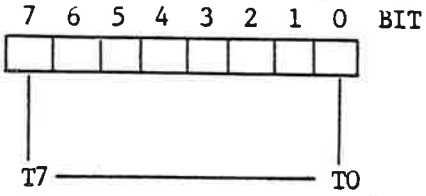
L0 through L7 are the lower 8 bits of the X-Y length which is the 10 bit binary value of the change in the X coordinate or the change in the Y coordinate, whichever is greater. The two most significant bits are placed in the Start Register.

$$IF|\Delta X| \geq |\Delta Y| \quad THEN \quad L = |\Delta X|$$

$$IF|\Delta X| < |\Delta Y| \quad THEN \quad L = |\Delta Y|$$

4.7 TEXTURE REGISTER:

WRITE ONLY

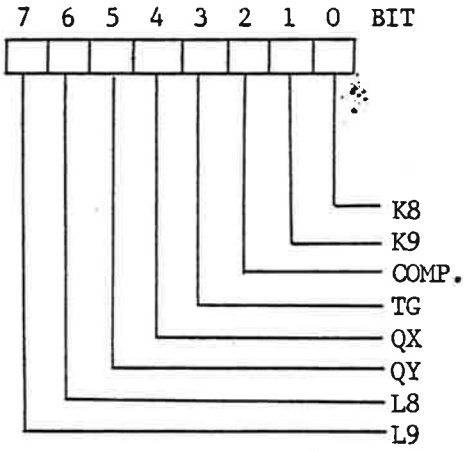


BASE ADDRESS + 06H = ☐
AS-SHIPPED ADDRESS = 66H

T0 through T7 determine the texture of the vector. A vector is composed of 8-pixel recurring segments where the pixels within the segments are drawn or not drawn depending on the state of the corresponding bit in the Texture Register. The pixel corresponding to bit 0 is the first pixel in the sequence. Pixels are drawn if the corresponding bit is one and they are not drawn if the corresponding bit is zero.

4.8 START REGISTER:

WRITE ONLY



LOC

BASE ADDRESS + 07H = [ ]

AS-SHIPPED ADDRESS = 67H

BITS 0 AND 1: K8 AND K9. These two bits are the two most significant bits of the vector slope (K). See section 4.5.

BIT 2: COMP. When a 1 is written to this bit, the Texture Register is complemented after each 8 pixel vector segment is drawn. For example, if the Texture Register initially contained all zero's, the vector would be drawn with the first 8 pixels not displayed, the next 8 pixels displayed, the next 8 pixels not displayed and so on. When this bit is 0, it has no effect.

BIT 3: TG. A one must be written to this bit when the vector slope is equal to or greater than 45° from the X axis. A zero must be written to this bit when the vector slope is less than 45° from the X axis.

BITS 4 AND 5: QX AND QY. These two bits must be used to indicate which quadrant the vector is to be drawn in, as shown in the following table.

QY	0	0	1	1
QX	0	1	0	1
QUADRANT	UPPER RIGHT	UPPER LEFT	LOWER RIGHT	LOWER LEFT

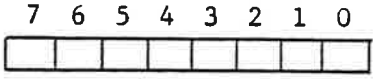
BITS 6 AND 7: L8 AND L9. These are the two most significant bits of the X-Y length. See section 4.6.

Writing to this register initiates the vector draw operation. Because of this, it must be accessed after the Vector Slope, X-Y length, and Texture Registers have been filled with the required parameters.

4.9 DATA PORTS: The RGB-Alpha must be isolated from the VAF board before writing to the following data ports. This may be done by setting OVERLAYEN (Mode Register-Bit 4) to zero.

4.9.1 RED DATA PORT:

READ/WRITE



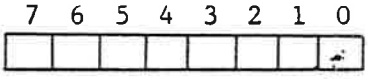
LOC

BASE ADDRESS + 08H = [ ]
AS-SHIPPED ADDRESS = 68H

This location is an 8 bit read/write port to the red look-up table location addressed by the current contents of the Address Register.

4.9.2 BLUE DATA PORT:

READ/WRITE



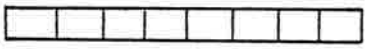
LOC

BASE ADDRESS + 09H = [ ]
AS-SHIPPED ADDRESS = 69H

This location is an 8 bit read/write port to the blue look-up table location addressed by the current contents of the Address Register.

4.9.3 GREEN DATA PORT:

READ/WRITE



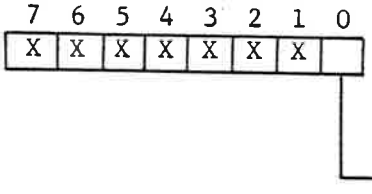
LOC

BASE ADDRESS + 0AH = [ ]
AS-SHIPPED ADDRESS = 6AH

This location is an 8 bit read/write port to the green look-up table location addressed by the current contents of the Address Register.

4.10 STATUS REGISTER:

READ ONLY



LOC

BASE ADDRESS + 0BH = [ ]
AS-SHIPPED ADDRESS = 6BH

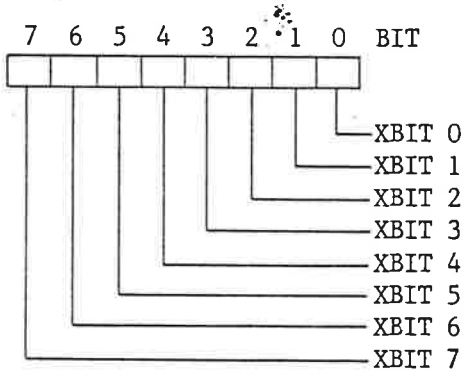
BIT 0: BUSY. When this bit is 1, the Vector Generator is in the process of drawing a vector. When this bit is 0, the Vector Generator is idle. Reading this register will reset the interrupt request flip-flop.

* * * * *

4.3 REGISTERS:

4.3.1 X-REGISTER LOW:

WRITE ONLY

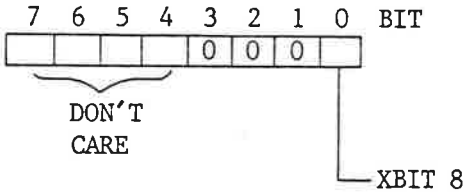


LOC.
BASE ADDRESS + 00H = [ ]
AS SHIPPED ADDRESS = A0H

This register holds the lower 8 bits of the X coordinate of the display memory location that is to be accessed.

4.3.2 X-REGISTER HIGH:

WRITE ONLY.

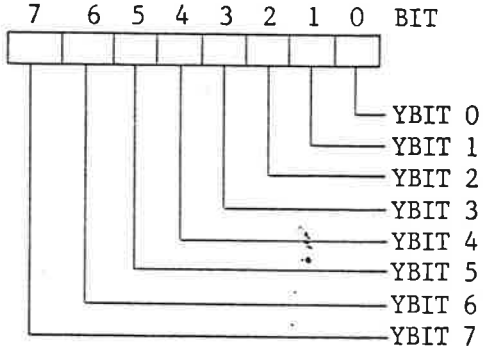


LOC.
BASE ADDRESS + 01H = [ ]
AS-SHIPPED ADDRESS = A1H

This register holds the 9th bit of the X coordinate for the RGB-GRAPH/32 and the RGB-GRAPH/64. If a One is written to any of bits 1-3 the clipping circuit will interdict memory access. This also applies to Bit 0 on the RGB-GRAPH/16.

4.3.3 Y-REGISTER LOW:

WRITE ONLY

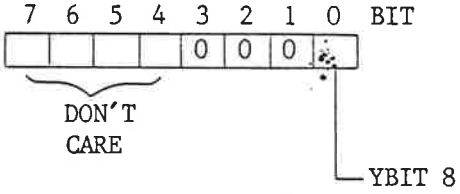


LOC.
BASE ADDRESS + 02H = [ ]
AS-SHIPPED ADDRESS = A2H

This register holds the lower 8 bits of the Y coordinate of the display memory location that is to be accessed.

4.3.4 Y-REGISTER HIGH:

WRITE ONLY

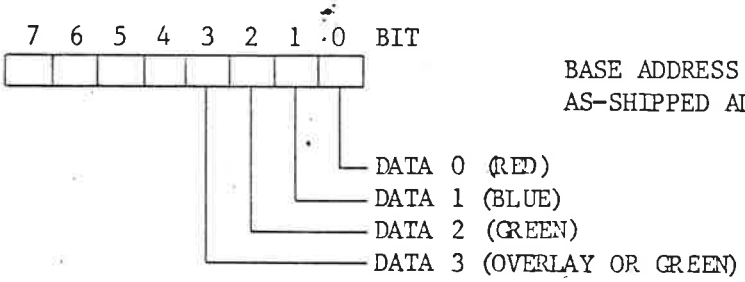


BASE ADDRESS + 03H = LOC.
AS-SHIPPED ADDRESS = A3H

This register holds the 9th bit of the Y coordinate for the RGB-GRAPH/64. If a One is written to any of Bits 1-3 the clipping circuit will interdict memory access. This also applies to Bit 0 on the RGB-GRAPH/16 and RGB-GRAPH/32.

4.3.5 DATA REGISTER LOW:

READ/WRITE



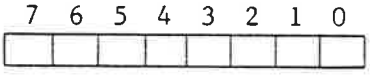
BASE ADDRESS + 04H = LOC.
AS-SHIPPED ADDRESS = A4H

As Shipped Configuration

Any four bits of this register can be used as a data port to the Display Memory if the appropriate straps are installed (see Section 5.2). When the board is shipped, straps are installed which allow the display memory to be accessed through the four least significant bits of this register, as shown above. Bits not selected for use are don't care for write and zero for read.

4.3.6 DATA REGISTER HIGH:

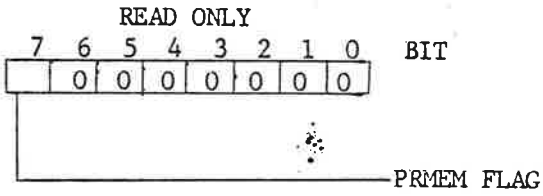
READ/WRITE



BASE ADDRESS + 05H = LOC.
AS-SHIPPED ADDRESS = A5H

As in the case of the Data Register Low, any four bits of this register can be used as a data port to the Display Memory if the appropriate straps are installed (see Section 5-3). Any bits not selected are "don't care" for write and zero for read.

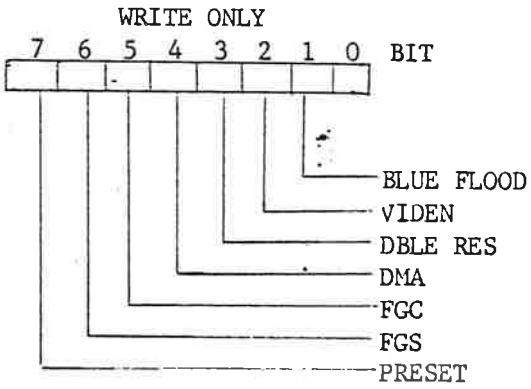
4.3.7 STATUS REGISTER:



BASE ADDRESS + 06H = LOC.
AS-SHIPPED ADDRESS = A6H

Bit 7: PRMEM FLAG. When this bit is one, the memory is being preset or a frame grab is in process. When this bit is zero, the memory is neither being preset nor is a frame grab in process.

4.3.8 CONTROL REGISTER NO. 1



BASE ADDRESS + 06H = LOC.
AS-SHIPPED ADDRESS = A6H

BIT 1: BLUE FLOOD. When this bit is zero, operation is normal. When this bit is one, the Blue output is driven on during active video. BLUE FLOOD is used to provide a visible raster to trigger the light pen. When activated, it is only generated on the composite video output; the TTL output are not affected.

Bit 2: VIDEN. When this bit is zero, TTL video is enabled. When this bit is one, TTL video is in high impedence (tri-state) mode.

Bit 3: DBLE RES. When this bit is one, bit planes 0 and 2 are multiplexed together and bit planes 1 and 3 are multiplexed together to provide twice the X-AXIS resolution. Note however, that the bits per pixel are halved. When this bit is zero, resolution is normal and there are four independent bit planes.

Bit 4: DMA. When this bit is one, the Display Memory can be accessed by DMA. All DMA transfers are made at the same 1K block of system address space, the base address of which, is set by straps (see Section 5.4). This system address space is mapped into different areas of the Display Memory, before the block transfer, by loading the X and Y Registers with the transfer's Display Memory starting address minus one. As the transfer proceeds, the X and Y Registers are automatically incremented before each byte transfer. When several 1K blocks are sequentially transferred to or from contiguous Display Memory, the X-Y starting address need only be loaded before the first block transfer.

When bit 4 is zero the RGB-GRAPH's Display Memory is accessed normally and the X and Y registers must be loaded with a new set of coordinates before each byte or word transfer.

4.3.8 CONTROL REGISTER NO. 1 (Cont'd):

Bit 5: FGC. When this bit is 0, the RGB-GRAPH operates in continuous frame grab mode. If a frame grabber is connected, the board will continually grab and display sequential frames: in effect, it will display what the camera sees. When the bit is 1, the RGB-GRAPH will freeze the frame that was in the display memory at the time the bit changed state. The user can watch the action, then freeze it.

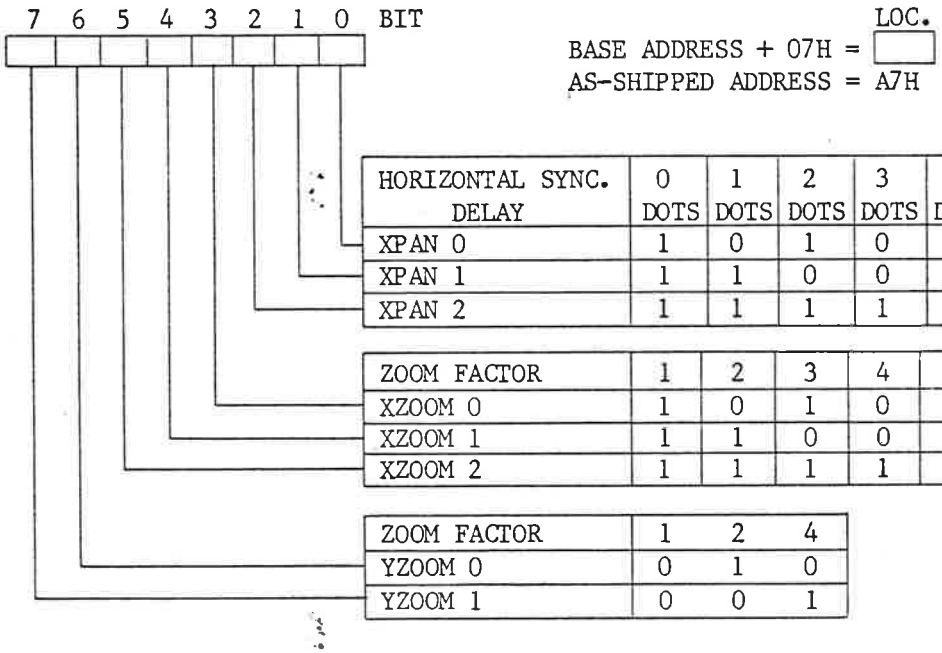
Bit 6: FGS. This bit is also provided for frame-grabbing operations. If a one is written to FGS , the RGB-GRAPH will grab and hold a single frame of video information. It will continue to display the information until one is again written to FGS, at which time a new frame will be grabbed. If a zero is written to this bit, it will have no effect.

Bit 7: PRESET. When a one is written to this bit, the Display Memory will be preset to the value in the Data Register. If a zero is written to this bit, it will have no effect.

NOTE: During initialization, the output operation to this register must be repeated twice. After initialization, one output is enough to load the register.

4.3.9 CONTROL REGISTER NO. 2:

WRITE ONLY



HORIZONTAL SYNC. DELAY	0 DOTS	1 DOTS	2 DOTS	3 DOTS	4 DOTS	5 DOTS	6 DOTS	7 DOTS
XPAN 0	1	0	1	0	1	0	1	0
XPAN 1	1	1	0	0	1	1	0	0
XPAN 2	1	1	1	1	0	0	0	0

ZOOM FACTOR	1	2	3	4	5	6	7	8
XZOOM 0	1	0	1	0	1	0	1	0
XZOOM 1	1	1	0	0	1	1	0	0
XZOOM 2	1	1	1	1	0	0	0	0

ZOOM FACTOR	1	2	4
YZOOM 0	0	1	0
YZOOM 1	0	0	1

NOTE: X zooms of greater than 4 are not possible when using the 256 x 256 format.

4.3.9 CONTROL REGISTER NO. 2 (Cont'd):

Bit 0-2: XPAN0-XPAN2. These three bits are used in conjunction with the CRTC starting address registers (R12 and R14) to horizontally pan the display. XPAN0-XPAN2 can be set to delay the horizontal sync. pulse by 1 through 7 dots (see Table above). A pan is accomplished by sequentially incrementing this delay until it reaches 7 dots then resetting XPAN0-XPAN2 and incrementing the CRTC starting address registers during vertical blanking. This operation is repeated at a rate that will give the required pan speed.

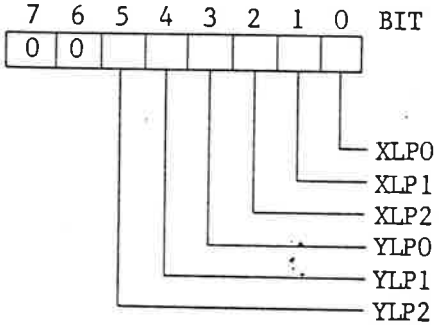
Bit 3-5: XZOOM0-XZOOM2. These three bits are used in conjunction with several CRTC registers (R0, R1, R2, R3, R12, R13) to expand the display along the horizontal axis (see Section 4.4).

Bit 6 & 7: YZOOM0 and YZOOM1. These two bits are used in conjunction with several CRTC registers (R3, R4, R5, R6, R7, R9, R12, R13) to expand the display along the vertical axis (see Section 4.4).

NOTE: For a normal display, X and Y zoom factors of one must be loaded.

4.3.10 AUXILIARY LIGHT PEN REGISTER:

READ ONLY



BASE ADDRESS + 08H = LOC.

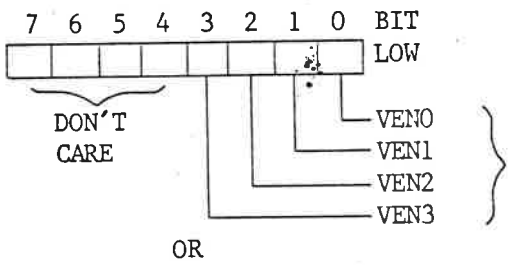
AS-SHIPPED ADDRESS = A8H

Bits 0-2: XLP0-XLP2. These bits are the three least significant bits of the light pen X coordinate. The most significant bits are provided by CRTC R17: bits 0-4 represent XLP3-XLP7 for 256 x 256 formats and bits 0-5 represent XLP3-XLP8 for 512 x 512 formats. Note that the data from CRTC R17 must be shifted to the left three bits before they can be combined with XLP0-XLP2 from this register. Unused bits in CRTC R17 are zero.

Bits 3-5: YLP0-YLP2. These bits are the three least significant bits of the light pen Y coordinate. The most significant digits are provided by CRTC R16: bits 0-4 represent YLP3-YLP7 for 256 x 256 formats and bits 0-5 represent YLP3-YLP8 for 512 x 512 formats. Note that the data from CRTC R16 must be shifted three spaces to the left before it can be combined with YLP0-YLP2 from this register. Unused bits in CRTC R16 are zero.

4.3.11 CONTROL REGISTER NO. 3:

WRITE ONLY

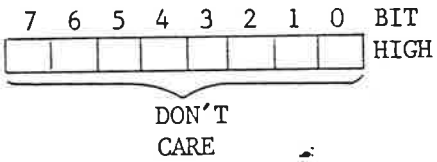


LOC.
BASE ADDRESS + 08H = [ ]
AS-SHIPPED ADDRESS = A8H

As Shipped Configuration

OR

WRITE ONLY



LOC.
BASE ADDRESS + 09H = [ ]
AS-SHIPPED ADDRESS = A9H

The four video enable bits can be strapped to any four bits in the two I/O locations shown above (see Section 5.4), and must be strapped to the same four bits that are used by the data port (see Section 4.3.5). The as-shipped configuration is shown here.

Bit 0: VEN0. When this bit is one, video from bit plane 0 is enabled. When this bit is zero, video from bit plane 0 is disabled.

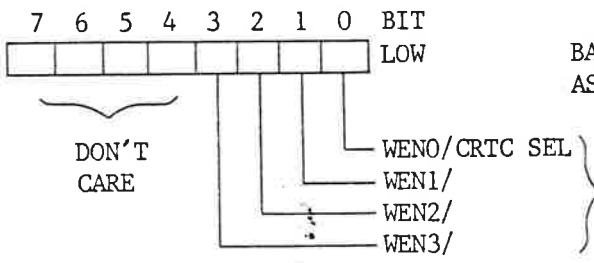
Bit 1: VEN1. When this bit is one, video from bit plane 1 is enabled. When this bit is zero, video from bit plane 1 is disabled.

Bit 2: VEN2. When this bit is one, video from bit plane 2 is enabled. When this bit is zero, video from bit plane 2 is disabled.

Bit 3: VEN3. When this bit is one, video from bit plane 3 is enabled. When this bit is zero, video from bit plane 3 is disabled.

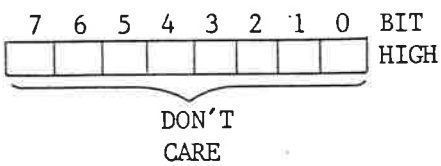
4.3.12 CONTROL REGISTER NO. 4:

WRITE ONLY



LOC.
BASE ADDRESS + 0AH = [ ]
AS-SHIPPED ADDRESS = AAH

As Shipped Configuration



LOC.
BASE ADDRESS + 0BH = [ ]
AS-SHIPPED ADDRESS = ABH

4.3.12 CONTROL REGISTER NO. 4 (Cont'd):

The four bits of Control Register No. 3 can be strapped to any four bits in the two I/O locations shown above (see Section 5.4) and must be strapped to the same four bits that are used as the data port. The as-shipped configuration is shown here.

Bit 0: WEN0/CRTC SEL/. When this bit is zero, bit plane 0 and the CRTC can be written to. When this bit is one, bit plane 0 and the CRTC cannot be written to. The CRTC SEL/ function is required for master slave configurations where the CRTCs of two or more boards at the same address are programmed differently.

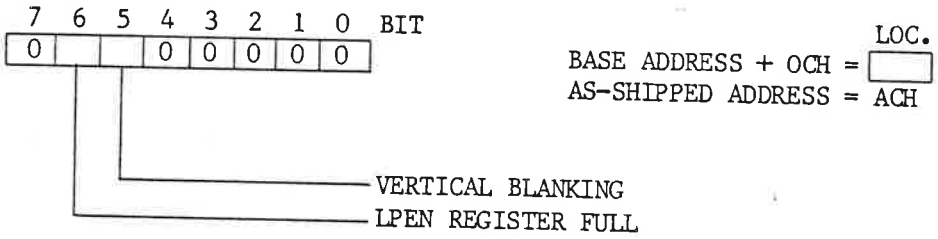
Bit 1: WEN1/. When this bit is zero, bit plane 1 can be written to. When this bit is one, bit plane 1 cannot be written to.

Bit 2: WEN2/. When this bit is zero, bit plane 2 can be written to. When this bit is one, bit plane 2 cannot be written to.

Bit 3: WEN3/. When this bit is zero, bit plane 3 can be written to. When this bit is one, bit plane 3 cannot be written to.

4.3.13 CRTC STATUS REGISTER:

READ ONLY



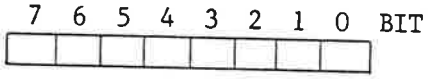
LOC.
BASE ADDRESS + 0CH = [ ]
AS-SHIPPED ADDRESS = ACH

Bit 5: VERTICAL BLANKING. When this bit is one, the scan is in vertical blanking. When this bit is zero, the scan is not in vertical blanking.

Bit 6: LPEN REGISTER FULL. This bit goes to one whenever a light pen strobe occurs. This bit goes to zero whenever either CRTC R16 or R17 are read.

4.3.14 CRTC ADDRESS REGISTER:

WRITE ONLY

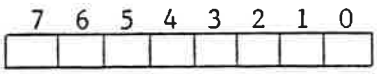


LOC.
BASE ADDRESS + 0CH = [ ]
AS-SHIPPED ADDRESS = ACH

When one of the CRTC registers is to be accessed, its address is placed in this register, then data is input or output through the CRTC Data Register. Addresses and descriptions for the CRTC registers are found in the CRTC data sheets.

4.3.15 CRTC DATA REGISTER:

READ/WRITE

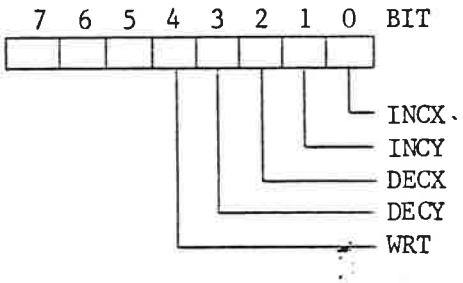


BASE ADDRESS + OEH = LOC.
AS-SHIPPED ADDRESS = AEH

This location is the data port to and from the CRTC.

4.3.16 VECTOR REGISTER:

WRITE ONLY



BASE ADDRESS + OFH = LOC.
AS-SHIPPED ADDRESS = AFH

BIT 0: INCX. When a one is written to this bit, the X-Register is incremented. Writing a zero to this bit has no effect on the X-Y coordinates.

Bit 1: INCY. When a one is written to this bit, the Y-Register is incremented. Writing a zero to this bit has no effect on the X-Y coordinates.

Bit 2: DECX. When a one is written to this bit, the X-Register is decremented. Writing a zero to this bit has no effect on the X-Y coordinates.

Bit 3: DECY. When a one is written to this bit, the Y-Register is decremented. Writing a zero to this bit has no effect on the X-Y coordinates.

Bit 4: WRT. When this bit is zero, the contents of the Data Register are automatically written to the Display Memory when the Vector Registor is loaded. When this bit is one, data is not automatically written to the Display Memory when the Vector Register is loaded.

NOTE: The Vector Register will not function properly if the RGB-GRAPH is in DMA mode.

Figure 5-1 shows the direction that the graphics trace will take when different values are written to the Vector Register.

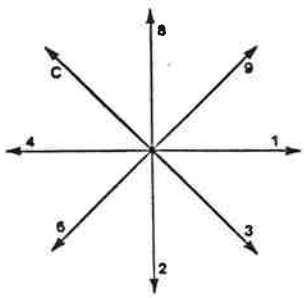


Figure 4.1 - VECTOR DIRECTION (BIT 0-3)